Floating-Point Error Analysis Using Symbolic Algebraic Computation

Mark P.W. Mutrie
Richard H. Bartels
Bruce W. Char

Research Report CS-87-08
February 1987

Floating-Point Error Analysis using Symbolic Algebraic Computation

*Mark P.W. Mutrie**
Richard H. Bartels
Bruce W. Char
Symbolic Computation Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, N2L 3G1

May 16, 1986

Abstract

The propagation of local roundoff errors must be studied to determine the stability in numerical algorithms. A floating-point error-analysis system is under development using the symbolic algebra language Maple [CGGW85]. This system can be used independently, to analyze the stability properties of formulae, or included as an intermediate step in an automatic code generation package.

Previous work done by Miller [Mil75], Larson and Sameh [LS78,LS80], and Stoutemyer [Sto77], among others, was considered in the development of this automated error-analysis system. The work is based on a general error-propagation model, a first-order Taylor series expansion of the accumulated rounding error with respect to the local rounding errors, and Bauer's computational graphs [Bau74]. The system accepts a sequence of expressions as input, identifies which data values in the input data range could lead to serious errors in the results of the given algorithm, and generates diagnostic information indicating the critical operations in the calculation. The identification of common subexpressions in the input formulae is used to make large problems more manageable. Future work includes extending the system to generate reformulations of the algorithm that minimize the accumulated roundoff error.

1 Introduction

In a numerical computation, local roundoff errors are introduced into the representation of the input data and into the result of each floating-point operation. The effect of these errors on the accumulated roundoff error, the difference between the computed result and the theoretical result,

*This work was funded under grants number A4076 and A5471 by the Natural Sciences and Engineering Research Council of Canada

may be either amplified or damped out as the computation proceeds. An analysis of the propagation of these errors through a complicated algorithm must be performed in order to determine its numerical stability, the sensitivity of the algorithm to roundoff error.

Since most algorithms involve numerous floating-point operations, it is not practical to perform a detailed error analysis by hand. However, error analysis techniques used to investigate the accumulated effect of rounding errors in numerical software can be automated. Furthermore, since the formulae and expressions used in many numerical computations are now derived using a symbolic algebra system [Wir80,GW84], error analysis procedures should also be an essential component of automatic code generation packages which act as an interface between symbolic algebra systems and numerical computing facilities.

Tools for performing floating-point roundoff error analysis are being developed in Maple with the goals of identifying which data values will lead to serious errors in the results of the given algorithm, providing diagnostic information indicating the critical operations in the calculation, and finally developing alternative formulations that avoid these critical operations.

In section 2, some of the previous developments towards automated error analysis are presented. Next, we look at the general error-propagation model [DB74,SB80] and computational graphs [Bau74,DM72], the framework around which we have built our system. This is followed by a description of the capabilities of the error analysis procedures in Maple. In closing, we consider how this work has extended the capabilities of earlier methods, and future work that can be undertaken.

2 Prior Work

2.1 Floating-Point Error Analysis

Miller [Mil75] has developed software that determines a value $\omega(x) \geq 0$ which is a measure of the effect of roundoff error in a numerical algorithm given an initial set of input data. This initial data set is the starting point of a heuristic analysis that uses numerical maximization procedures to search for data values that cause $w(d)$ to become unacceptably large, indicating that the algorithm could perform poorly. Since the method is heuristic, it may fail in finding a data set that causes a large value $\omega(d)$, even though one exists. Miller's system is based on strictly numerical techniques, and he uses absolute errors in his approach.

Miller describes techniques which allow certain floating-point operations to be considered error-free and ignored in an error analysis [Mil76]. This technique may be useful in extending our error analysis system in Maple to handle larger problems.

Larson and Sameh [LS78,LS80] have developed a method of error analysis that considers relative errors, which are more commonly used in studying floating-point arithmetic. They apply their method to to give an accurate numerical analysis of an algorithm for fixed data values. They have established a relationship between local and accumulated relative errors that gives rise to an underdetermined system of equations, which is then used in forward, backward, or B (two-sided or beidseitige) error analyses.

Their method is based on a computational-graph model of roundoff error propagation. The symbolic error analysis system under development in Maple uses a similar approach. However,

rather than applying the method numerically for a single set of input data, a parameterized data set is used to produce symbolic expressions for the accumulated error in the output.

Stummel [Stu80,Stu81] has also developed the theory for a forward error analysis of a numerical calculation using an approach similar to that of Larson and Sameh. He has applied these theoretical results to a series of algorithms, but has not automated the process.

Linnainmaa [Lin76] describes algorithms for computing the coefficients of the Taylor series expansion for the accumulated absolute error in terms of the local absolute errors. He gives algorithms for determining the first and second order coefficients. We use the first-order coefficients of the Taylor series expansion for the accumulated relative error, instead, which is described in section 3.

In general, numerical methods have the disadvantage that they can only be applied to one set of fixed input values at a time. A symbolic error analysis can provide information over the entire data range of the problem. The data values around which numerical stability is a problem can be generated for the user, and the critical operations leading to this instability can be identified. Numerical error analysis systems can, however, handle larger problems than symbolic systems.

2.2 Symbolic Error Analysis Systems

Stoutemyer [Sto77] developed a symbolic error analysis procedure in REDUCE which also applies the error propagation model based on computational graphs described in section 3.2. This approach takes advantage of the natural recursive properties of the language REDUCE (also available in other symbolic algebra languages). Stoutemyer's procedures accept a single formulae as input, and generate an expression that is a first-order bound on the accumulated error expressed in terms of the symbolic local roundoff errors. He also provides procedures based on a statistical error analysis model which calculate a bound and roundoff variance for the accumulated error in terms of the machine unit roundoff.

We use the same error propagation model as Stoutemyer, to generate a first-order expression for the accumulated error, but use a different implementation that allows us to consider more than one input expression. We also automatically identify the data values that cause the symbolic accumulated error bound to become large.

Hulshof and van Hulzen [HvH84] have taken a different approach. Their system, implemented in REDUCE, determines the precision required in each floating-point operation so that an overall error bound can be satisfied. The system implemented in Maple assumes all operations are performed with a fixed precision.

Having surveyed some of the work that has been done towards automating floating-point error analysis, let us now consider the error propagation model upon which the system being developed in Maple is based.

3 Error Propagation Model

3.1 Input Errors and Condition Numbers

The *general error-propagation model* is used to investigate the accumulated effect of local rounding errors which occur in numerical computations. Consider a problem with n input values $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in which we must find m output values $\mathbf{y} = (y_1, y_2, \ldots, y_m)$. An algorithm to

solve this problem can be represented as a multivariate vector function $\mathbf{y} = \varphi(\mathbf{x})$ where $y_i = \varphi_i(x_1, \ldots, x_n)$.

In order to develop the roundoff model, one usually begins by considering how input errors $\Delta\mathbf{x}$ of $\mathbf{x}$ are propagated through the algorithm assuming no further errors. Stoer and Bulrisch [SB80] give a complete development of this model which is summarized here. We can study the effect of the input errors by replacing the input data $\mathbf{x}$ by $\tilde{\mathbf{x}} = \mathbf{x} + \Delta\mathbf{x}$ in the algorithm, which will give the result $\tilde{\mathbf{y}} = \varphi(\tilde{\mathbf{x}})$ instead of $\mathbf{y} = \varphi(\mathbf{x})$. Consider just one component y_i of $\mathbf{y}$, $i = 1, \ldots, m$. Approximating $\Delta y_i = \tilde{y}_i - y_i$, using just the first order terms of the Taylor series expansion, gives

$$\Delta y_i = \varphi_i(\tilde{\mathbf{x}}) - \varphi_i(\mathbf{x}) = \sum_{j=1}^{n} \frac{\partial \varphi_i(\mathbf{x})}{\partial x_j}(\tilde{x}_j - x_j) = \sum_{j=1}^{n} \frac{\partial \varphi_i(\mathbf{x})}{\partial x_j}\Delta x_j.$$

The value $\frac{\partial \varphi_i(\mathbf{x})}{\partial x_j}$ indicates the sensitivity of y_i to absolute perturbations Δx_j in x_j. In matrix notation, the first order approximation for $\Delta\mathbf{y}$ can be written as, $\Delta\mathbf{y} = D\varphi(\mathbf{x})\Delta\mathbf{x}$, a product of $\Delta\mathbf{x}$ and the Jacobian matrix $D\varphi(\mathbf{x})$.

A similar error propagation formula can be derived for relative errors if $y_i \neq 0$ for $i = 1, \ldots, m$ and $x_j \neq 0$ for $j = 1, \ldots, n$. The relative error of $\tilde{x}_i$ is defined as $\varepsilon_{\tilde{x}_i} = \frac{\tilde{x}_i - x_i}{x_i}$. The relative error in the output component y_i is

$$\frac{\tilde{y}_i - y_i}{y_i} = \sum_{j=1}^{n} \left[\frac{x_j}{\varphi_i(\mathbf{x})} \frac{\partial \varphi_i(\mathbf{x})}{\partial x_j} \right] \varepsilon_{\tilde{x}_j} = \sum_{j=1}^{n} \frac{\rho\varphi_i(\mathbf{x})}{\rho x_j} \varepsilon_{\tilde{x}_j}$$

where $y_i = \varphi_i(\mathbf{x})$, $i = 1, \ldots, m$. ρ is used as the relative partial differential operator. The value $\frac{\rho\varphi_i(\mathbf{x})}{\rho x_j}$ is a relative partial derivative called the *condition number* or *amplification factor* that indicates how strongly the relative error in y_i is affected by the relative error in x_j.

Stoer and Bulrisch also describe the model which considers both error in the input data, and roundoff error occurring at each intermediate floating-point operation. This extended model can also be described using a graphical approach which is presented in section 3.2.

3.2 Computational Graphs

Much of the work in automating roundoff error analysis is based on a model for relative error propagation that can be described using computational graphs.

A computational graph of an algorithm is a directed graph which can be thought of as a parse tree of the algorithm. In this way, the algorithm can be expressed graphically in terms of its elementary operations.

Each node of a computational graph, ξ_i, $i = 1, \ldots, r$, represents an element of an operand set, $\{\xi_1, \xi_2, \ldots, \xi_n, \xi_{n+1}, \ldots, \xi_r\}$, which includes the n input values and $r-n$ computational values. The input values, x_i, are equivalent to ξ_i for $i = 1, \ldots, n$, and the computational values are represented by ξ_i, for $i = n+1, \ldots, r$. The output values are elements of this operand set: $y_j \in \{\xi_1, \xi_2, \ldots, \xi_r\}$ for $j = 1, \ldots, m$.

If the algorithm includes the elementary operation $\xi_i = \xi_j \odot \xi_k$, then directed arcs lead from nodes ξ_j and ξ_k to node ξ_i. This binary operation is represented graphically in Figure 1 (a) below. Other n-*ary* operations can be represented in a similar way.

Figure 1: (a) $\xi_i = \xi_j \odot \xi_k$. (b) Arcs labelled with condition numbers.

First, consider this elementary operation independent of the rest of the computation. The error in the result of any arithmetic operation is composed of:

1. the error due to errors in the operands, and
2. the roundoff error incurred in performing this operation.

If we ignore item 2, the roundoff error committed in executing this operation, we can apply the model developed in the previous section which assumes error in the input and exact operations. $\mathcal{A}(\xi_l)$ is used as a convenient way to represent the relative error in the node ξ_l. This corresponds to the relative error, $\varepsilon_{\tilde{x}_l}$, in $\tilde{x}_l$ in the general error-propagation model. Hence, the relative error in the output node, ξ_i, is

$$\mathcal{A}(\xi_i) = \frac{\rho\xi_i}{\rho\xi_j}\mathcal{A}(\xi_j) + \frac{\rho\xi_i}{\rho\xi_k}\mathcal{A}(\xi_k).$$

This expression can be represented in the computational graph by labelling the arcs with the corresponding condition numbers, as indicated in Figure 1 (b).

If we include the local rounding error incurred in performing this operation, which we represent as $\mathcal{L}(\xi_i)$, then

$$\mathcal{A}(\xi_i) = \frac{\rho\xi_i}{\rho\xi_j}\mathcal{A}(\xi_j) + \frac{\rho\xi_i}{\rho\xi_k}\mathcal{A}(\xi_k) + \mathcal{L}(\xi_i).$$

Note that $\mathcal{L}(\xi_i)$ is typically very small, being bounded by the unit roundoff error for the specified machine, and that $\mathcal{A}(\xi_l) = \mathcal{L}(\xi_l)$ for the input nodes.

Now, consider a complete algorithm represented by a computational graph. The error propagation can be represented by labelling every directed arc in the computational graph with the local relative differential condition number of the elementary operation represented by the arc and the associated nodes.

The form of the relative differential condition number for each arithmetic operation, and the standard transcendental functions can be determined using the standard rules of differentiation. Then, for a given elementary operation, one needs only to substitute the values of the operands and the result into the appropriate formula.

For example, consider the computational graph of the algorithm

$$\begin{aligned} y_1 &= b \cdot a - a \cdot c \\ y_2 &= 1.0 - \cos(d) \end{aligned}$$

given in Figure 2.

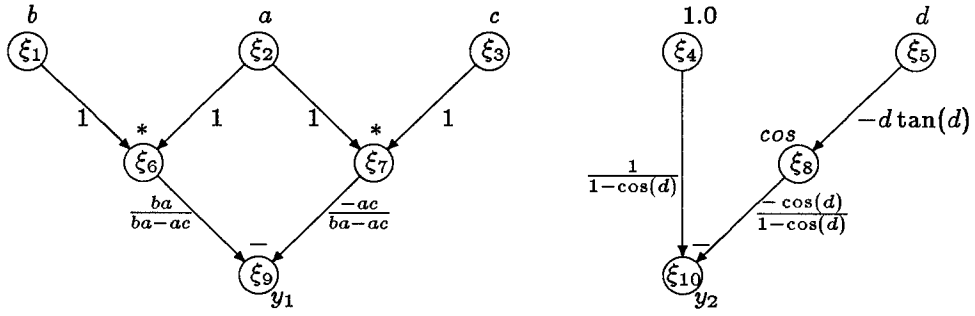

Figure 2: Computational graph of y_1 and y_2

Starting at the input nodes, the local relative error in each node, $\mathcal{L}(\xi_i)$, subsequently affects the value in another node if there is a path connecting them. The effect of all *previous* local relative errors, and the local relative error at this node, combine to produce a total or accumulated relative error, $\mathcal{A}(\xi_i)$, in each node. We would like to express the accumulated relative error in each node in terms of the local relative errors.

For the path between nodes ξ_k and ξ_i, $k < i$, we can define a *path product* as the product of the local relative differential condition numbers on the directed arcs of this path. The value of the accumulated relative error in a node ξ_i, $\mathcal{A}(\xi_i)$, is given by

$$\mathcal{A}(\xi_i) = \sum_{k=1}^{r} P_k \mathcal{L}(\xi_k)$$

where P_k is the sum of all path products for the paths leading from node ξ_k to ξ_i. $P_i = 1$ since the local error incurred at node ξ_i has not been propagated.

This corresponds to computing the relative partial differential $\rho\xi_i$ by applying the chain rule for relative partial derivatives, and then adding on a term for the local rounding error at this node.

This approach generates a system of m homogeneous equations in the unknowns $\mathcal{A}(\xi_i)$ where ξ_i is one of the output nodes, and $\mathcal{L}(\xi_l)$, the local relative roundoff error in each node, $l = 1, \ldots, r$. As an example of an equation that one would find in this system, consider the equation for the error in the output node ξ_9:

$$0 = -\mathcal{A}(\xi_9) + \frac{ba}{ba-ac}\mathcal{L}(\xi_1) + 1\mathcal{L}(\xi_2) + \frac{-ac}{ba-ac}\mathcal{L}(\xi_3) + \frac{ba}{ba-ac}\mathcal{L}(\xi_6) + \frac{-ac}{ba-ac}\mathcal{L}(\xi_7) + 1\mathcal{L}(\xi_9).$$

Note that this constitutes an expression for $\mathcal{A}(\xi_9)$ in terms of the $\mathcal{L}(\xi_l)$ and that we must first calculate the path products in order set up such an equation.

Alternatively, we can express the accumulated relative error in each node *recursively* in terms of the accumulated relative error in the operands plus any local rounding error at this node. Using this approach, we obtain an underdetermined system of $r - n$ homogeneous equations, one for each computational value, in the unknowns $\mathcal{A}(\xi_i)$, $i = n+1, \ldots, r$ and $\mathcal{L}(\xi_l)$, $l = 1, \ldots, r$. Each equation has the form:

$$0 = -\mathcal{A}(\xi_i) + \frac{\rho\xi_i}{\rho\xi_j}\mathcal{A}(\xi_j) + \frac{\rho\xi_i}{\rho\xi_k}\mathcal{A}(\xi_k) + \mathcal{L}(\xi_i)$$

where the number of terms on the right hand side depends on the type of operator – the form given assumes a binary operator. As an example of an equation that one would find in this system, again consider the equation for the error in the output node ξ_9:

$$0 = -\mathcal{A}(\xi_9) + \frac{ba}{ba - ac}\mathcal{A}(\xi_6) + \frac{-ac}{ba - ac}\mathcal{A}(\xi_7) + 1\mathcal{L}(\xi_9).$$

Each of the two approaches give rise to a system of equations which relate the local and the accumulated relative errors. Larson and Sameh [LS78] have shown that the two approaches are equivalent, but that the second approach produces a system of equations that is very sparse, with less than or equal to 4 elements per row, which can be solved very efficiently to generate the expressions for the $\mathcal{A}(\xi_i)$ in terms of the $\mathcal{L}(\xi_l)$, as above, without first having to compute path products.

By studying the coefficients in the expressions for the accumulated roundoff error (the accumulated condition numbers), one can determine which local roundoff errors have the greatest effect upon the accumulated error of the result [Sto77, page 29]. Depending on the value of a condition number, the effect of the local rounding errors can be either amplified or damped out. We will come back to this idea in section 4 on automated error analysis in Maple.

4 Automated Error Analysis Using Maple

Stoutemyer describes some of the features that an "ideal automated error analysis procedure" should include [Sto77, page 40]. Procedures are being developed in Maple based on the error-propagation models just presented, and using the description of the ideal as a guide. The components of the ideal system will be brought to light as we discuss the automated error analysis procedure.

4.1 The Input

Since symbolic algebra systems are used to generate many of the formulae used in numerical computations, it is natural to want the error analysis tools to be part of a symbolic/numeric interface package which automatically generates code in a numeric programming language. The error analysis procedure should accept as input one or more of these expressions.

The Maple procedures accept a list of expressions, that can be thought of as a segment of code to be inserted into a numerical routine. For example, the input for the example in section 3.2 would have the form:

$$l1 := [y_1 = a * b - a * c, y_2 = 1.0 - cos(d)].$$

Alternatively, one could design the procedures to be able to accept as input a fragment of code or an entire numerical computation program written in a language such as FORTRAN. This involves handling the analysis of expressions for which the number or sequence of operations depends upon the data. The system must handle code that contains a segment that iterates until some convergence criterion is satisfied, or that involves a conditional branch which selects alternative subexpressions depending on the outcome of the data-dependent test, or performs operations on N by N matrices where N is a parameter [Sto77, page 40]. The extensions to handle data-dependent code have not yet been considered in the Maple system.

4.2 Common Subexpressions

The tools for automated error analysis should be able to identify common subexpressions to reduce the complexity of the error analysis of a large problem and to facilitate problem reformulation. Identifying common subexpressions becomes an important part of an error analysis, because it may make the difference between being able to handle the problem or not.

M. Monagan has developed an efficient optimize procedure[1] for Maple which relies on the fact that Maple automatically finds many common subexpressions in building the internal representation of an expression. As an alternative search technique for common subexpressions, van Hulzen's approach [vH81,vH83] can be applied. The former method has the advantage that the structure of the expression is not altered so that numerical properties are not affected.

The search for common subexpression introduces temporary variables into the formulae to represent the common subcomponents. Since the error analysis procedure is able to handle a sequence of expressions, the new assignments representing the common subcomponents are added to the list of input expressions in the appropriate order.

Stoutemyer points out that Bauer [Bau74] and Miller [Mil76] have described methods which permit some of the errors due to floating-point operations to be neglected in an error analysis. Miller has shown that by using graph transformations, a rounding error in one floating-point operation can be reassigned to other operations in some cases without altering the final result of the error analysis. In these cases, certain operations can be considered error-free, and can be ignored in the analysis. These ideas could be used in conjunction with a common subexpression search in reducing the complexity of the error analysis. Further comments on common subexpressions, including mention of some difficulties, appear below in Section 4.5.

4.3 Calculating the Accumulated Condition Numbers

At this point in the analysis we have a list of one or more expressions that represent formulae, or pieces of formulae to analyze. Using the recursive formulation for the accumulated roundoff error in each computational value, a matrix $\tilde{M}$ containing the local relative differential condition numbers is generated.

For the computational graph in Figure 2 we have

$$\begin{bmatrix} \mathcal{A}(\xi_6) \\ \mathcal{A}(\xi_7) \\ \mathcal{A}(\xi_8) \\ \mathcal{A}(\xi_9) \\ \mathcal{A}(\xi_{10}) \end{bmatrix} = \begin{bmatrix} 0 & & & & \\ & 0 & & & \\ \frac{ba}{ba-ac} & \frac{-ac}{ba-ac} & 0 & & \\ & & \frac{-\cos(d)}{1-\cos(d)} & 0 & \\ & & & & 0 \end{bmatrix} \begin{bmatrix} \mathcal{A}(\xi_6) \\ \mathcal{A}(\xi_7) \\ \mathcal{A}(\xi_8) \\ \mathcal{A}(\xi_9) \\ \mathcal{A}(\xi_{10}) \end{bmatrix} +$$

$$\begin{bmatrix} 1 & 1 & & & \\ & 1 & 1 & & \\ & & 0 & & -d\tan(d) \\ & & & 0 & \\ & & & \frac{1}{1-\cos(d)} & 0 \end{bmatrix} \begin{bmatrix} \mathcal{L}(\xi_1) \\ \mathcal{L}(\xi_2) \\ \mathcal{L}(\xi_3) \\ \mathcal{L}(\xi_4) \\ \mathcal{L}(\xi_5) \end{bmatrix} + \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} \mathcal{L}(\xi_6) \\ \mathcal{L}(\xi_7) \\ \mathcal{L}(\xi_8) \\ \mathcal{L}(\xi_9) \\ \mathcal{L}(\xi_{10}) \end{bmatrix}$$

which we write as $\tilde{u} = \tilde{B} \cdot \tilde{u} + \tilde{A} \cdot \tilde{v} + \tilde{I} \cdot \tilde{w}$. We can express this vector equation as a system of

[1] Information on this procedure will be available in the next edition of the Maple User's Guide

homogeneous equations

$$\left[\begin{array}{c|c|c} \tilde{B}-I & \tilde{A} & I \end{array}\right]\left[\begin{array}{c} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{array}\right]=\left[\begin{array}{c} 0 \end{array}\right]$$

which we write as $\tilde{M}\tilde{z}=0$.

We want to generate a homogeneous system of equations, $\bar{M}\bar{z}=0$, expressing the accumulated relative error in the output values just in terms of the local relative errors, $\mathcal{L}(\xi_l)$, $l=1,\ldots,r$. If the jth row of $\tilde{M}$ describes the accumulated roundoff error for an output variable, we can generate a corresponding row i in $\bar{M}$ by using the relationship between the systems $\tilde{M}\tilde{z}=0$ and $\bar{M}\bar{z}=0$ established by Larson and Sameh [LS78]. If we define $\tilde{C}=[\,\tilde{A}\mid\tilde{B}-I\,]$ and $\bar{C}=[\,\bar{A}\mid\bar{B}\,]$, then the ith row of $\bar{C}$ which corresponds to the jth row of $\tilde{C}$ is given by the system

$$\left[\begin{array}{cc} I & -\tilde{A}^t \\ & -(\tilde{B}-I)^t \end{array}\right]\left[\begin{array}{c} \bar{c}_i \end{array}\right]=\left[\begin{array}{c} e_{n+j} \end{array}\right]$$

where e_{n+j} is the $(n+j)th$ column of the identity matrix. This is a sparse upper triangular matrix system with at most 4 elements in each row which can be solved very efficiently using the outer-product form for solving triangular systems described by George and Liu [GL81, pages 25–28].

For the given problem, we generate

$$\bar{C}=\left[\begin{array}{ccccc|ccccc} \frac{ba}{ba-ac} & 1 & \frac{-ac}{ba-ac} & & & \frac{ba}{ba-ac} & \frac{-ac}{ba-ac} & & 1 & \\ & & & \frac{1}{1-\cos(d)} & \frac{d\sin(d)}{1-\cos(d)} & & & \frac{-\cos(d)}{1-\cos(d)} & & 1 \end{array}\right]$$

where $\bar{u}=\bar{A}\cdot\bar{v}+\bar{B}\cdot\bar{w}$ is

$$\left[\begin{array}{c} \mathcal{A}(\xi_9) \\ \mathcal{A}(\xi_{10}) \end{array}\right]=\left[\begin{array}{ccccc} \frac{ba}{ba-ac} & 1 & \frac{-ac}{ba-ac} & & \\ & & & \frac{1}{1-\cos(d)} & \frac{d\sin(d)}{1-\cos(d)} \end{array}\right]\left[\begin{array}{c} \mathcal{L}(\xi_1) \\ \mathcal{L}(\xi_2) \\ \mathcal{L}(\xi_3) \\ \mathcal{L}(\xi_4) \\ \mathcal{L}(\xi_5) \end{array}\right]$$

$$+\left[\begin{array}{ccccc} \frac{ba}{ba-ac} & \frac{-ac}{ba-ac} & & 1 & \\ & & \frac{-\cos(d)}{1-\cos(d)} & & 1 \end{array}\right]\left[\begin{array}{c} \mathcal{L}(\xi_6) \\ \mathcal{L}(\xi_7) \\ \mathcal{L}(\xi_8) \\ \mathcal{L}(\xi_9) \\ \mathcal{L}(\xi_{10}) \end{array}\right]$$

Again we can think of this as a system of homogeneous equations

$$\left[\begin{array}{c|c|c} -I & \bar{A} & \bar{B} \end{array}\right]\left[\begin{array}{c} \bar{u} \\ \bar{v} \\ \bar{w} \end{array}\right]=\left[\begin{array}{c} 0 \end{array}\right]$$

which we write as $\bar{M}\bar{z}=0$. The system of equations, $\bar{M}\bar{z}=0$, describes the accumulated roundoff errors in terms of the local roundoff errors. For analyzing the accumulated condition numbers, it is convenient if we think of the system $\bar{M}\bar{z}=0$ in the form $\bar{u}=\bar{A}\cdot\bar{v}+\bar{B}\cdot\bar{w}$.

4.4 Analyzing the Condition Numbers

The elements of a row of $\bar{A}$ represent the coefficients of $\mathcal{L}(\xi_i)$, $i = 1, \ldots, n$, the local relative error in the input nodes. These are the condition numbers for the *inherent error*, the error due to inexact data or data rounded on input. The elements of the matrix $\bar{B}$ are the coefficients of $\mathcal{L}(\xi_i)$, $i = n+1, \ldots, r$, the local relative error in the nodes of the computational values. These are the condition numbers for the *generated error*, the error generated by the propagation of roundoff error through the algorithm due to inexact floating-point arithmetic.

In matrix notation the relative error bound is

$$\begin{aligned} \|\bar{u}\|_\infty &\leq \|\bar{A}\|_\infty \cdot \|\bar{v}\|_\infty + \|\bar{B}\|_\infty \cdot \|\bar{w}\|_\infty \\ &\leq \|\bar{A}\|_\infty + \|\bar{B}\|_\infty \end{aligned}$$

This essentially constitutes a forward error analysis. We have a first-order bound on the accumulated error in terms of a parameterized set of input values.

The *solve* and *fsolve* and *limit* procedures of Maple are being used to determine the singularities in the condition numbers. This provides us with information on what data values cause a large accumulated error bound. The node associated with a condition number that has a singularity is used to identify a critical operation in the problem.

For the given example, the procedures identify that numerical difficulties could occur when $a = 0$, or $b = c$ in the expression y_1, and when $d = 0$ in y_2. The subtraction operation for each expression is identified as the critical operation.

A backward error analysis is not always possible. As Stoutemyer [Sto77, page 40] and Larson and Sameh [LS80, page 286–287] point out, if a data value occurs in several roles in an algorithm, then it is often difficult, and sometimes impossible to associate the same error value, say δ, with it for each occurrence. Larson has attempted to overcome this problem by using a B (two-sided or beidseitige) analysis. At this time, only a forward error analysis has been implemented in Maple.

4.5 Handling Common Subexpressions

In order to handle the common subexpressions, the groundwork developed by Larson and Sameh for composite algorithms is used [LS80, pages 293–296]. A composite algorithm is one which is made up of smaller sub-algorithms where the outputs of one sub-algorithm serve as the inputs to the next. A common subexpression can be thought of as a sub-algorithm. Larson and Sameh show that for a composite algorithm, algorithm 3, composed of algorithm 1 followed by algorithm 2,

$$\bar{A}_3 = \bar{A}_1 \cdot \bar{A}_2$$

and

$$\bar{B}_3 = \left[\bar{A}_2 \cdot \bar{B}_1, \bar{B}_2\right]$$

A forward error analysis of the problem may be undertaken using these formula. For the inherent error we have the bound

$$\|\bar{A}_3\|_\infty \leq \|\bar{A}_1\|_\infty \cdot \|\bar{A}_2\|_\infty$$

and for the generated error

$$\|\bar{B}_3\|_\infty \leq \|\bar{A}_2\|_\infty \cdot \|\bar{B}_1\|_\infty + \|\bar{B}_2\|_\infty$$

These bounds are not as tight as those that would be obtained by actually calculating $\bar{A}_3$ and $\bar{B}_3$.

In a symbolic context, the problem is complicated by the fact that algorithm 2 contains temporary variables representing the common subexpressions. In order to decide where singularities occur in the condition numbers (and hence to determine where the bounds become large) it is necessary to first find the singularities in the condition numbers of A_1 and B_1, and then use these values when solving for the condition numbers of the second algorithm. If many common subexpressions are involved, not only does the number of different combinations of possible substitutions into the condition numbers of the second algorithm become prohibitively large, but also one must consider the singularities that occur in the composition of the subexpressions. Further work is required to try to overcome some of these difficulties.

4.6 Algorithm Reformulation

It would be very helpful if the error analysis procedures could generate alternative formulations of the expressions which minimize the accumulated roundoff error. The reformulation of an algorithm into a more stable form does not change the inherent error, the error due to inexact data or data rounded on input, associated with the coefficients of the matrix $\bar{A}$. The reformulation of the algorithm *does* effect the generated error, the error generated by the propagation of roundoff error through the algorithm due to inexact floating-point arithmetic, associated with the matrix $\bar{B}$.

Knowing what data values cause the bound on the generated error to become large, and having identified the critical parts of the computations, the error analysis system could attempt to reformulate the expression(s). The strategies for reformulating algorithms could be based on a set of simple rules which guide the system towards that formulation which minimizes the accumulated roundoff error. The system may suggest different formulations for different ranges of data. Further investigation of these ideas is required.

5 Concluding Remarks

The goal of the work that we have presented is to extend the capabilities of automated floating-point error analysis. We have built a system around a error-propagation model that can handle a sequence of expressions that could represent segment of an algorithm; handling more general algorithms is a worthwhile goal to work towards. We have also automated the step of analyzing the accumulated condition numbers. As well, the capability to perform a common subexpression search to make large problems more manageable has been incorporated into the system. Techniques for generating alternative formulations of an algorithm that minimizes the accumulated roundoff error is another problem that needs to be investigated.

The goal of an automatic code generation system is to produce a program in a numerical programming language that we are confident is correct, efficiently expressed, and *numerically stable.* The procedures that have been presented here can be used to help achieve this goal.

References

[Bau74] F. L. Bauer. Computational graphs and rounding error. *SIAM Journal of Numerical Analysis*, 11(1):87–96, March 1974.

[CGGW85] B. W. Char, K. O. Geddes, G. H. Gonnet, and S. M. Watt. *Maple User's Guide.* WATCOM Publications Ltd., Waterloo, Ontario, Canada, 4 edition, 1985.

[DB74] G. Dahlquist and Å. Björck. *Numerical Methods*, chapter 2, pages 21–59. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1974.

[DM72] W. S. Dorn and D. D. McCracken. *Numerical Methods with Fortran IV Case Studies*, chapter 2, pages 65–98. John Wiley and Sons, Inc., New York, 1972.

[GL81] Alan George and Joseph W.-H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1981.

[GW84] Barbara L. Gates and Paul S. Wang. A LISP-Based Ratfor Code Generator. In V. Ellen Golden, editor, *Proceedings of the 1984 Macsyma User's Conference*, pages 319–329, General Electric, Schenectedy, New York, July 1984.

[HvH84] B. J. A. Hulshof and J. A. van Hulzen. Automatic error cumulation control. In J. Fitch, editor, *Eurosam 84, International Symposium on Symbolic and Algebraic Computation, Cambridge, England, July 9-11*, pages 260–271, Springer-Verlag, New York, 1984.

[Lin76] S. Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Nordisk Tidskrift for Informationsbehandling*, 16(1):146–160, 1976.

[LS78] J. L. Larson and A. H. Sameh. Efficient calculation of the effects of roundoff errors. *ACM Transactions on Mathematical Software*, 4(3):228–236, September 1978.

[LS80] J. L. Larson and A. H. Sameh. Algorithms for roundoff error analysis – a relative error approach. *Computing*, 24(4):275–297, 1980.

[Mil75] W. Miller. Software for roundoff analysis. *ACM Transactions on Mathematical Software*, 1(2):108–128, 1975.

[Mil76] W. Miller. Graph transformations for roundoff analysis. *SIAM Journal of Computing*, 5:204–216, 1976.

[SB80] J. Stoer and R. Bulrisch. *Introduction to Numerical Analysis*, pages 5–27. Springer-Verlag, New York, 1980.

[Sto77] David R. Stoutemyer. Automatic error analysis using computer algebraic manipulation. *ACM Transactions on Mathematical Software*, 3(1):26–43, March 1977.

[Stu80] F. Stummel. *Rounding Error Analysis of Elementary Numerical Algorithms*, pages 169–195. Springer-Verlag, 1980.

[Stu81] F. Stummel. Perturbation theory for evaluation algorithms of arithmetic expressions. *Mathematics of Computation*, 37(156):435–473, 1981.

[vH81] J. A. van Hulzen. Breuer's grow factor algorithm in computer algebra. In *Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation*, pages 100–104, 1981.

[vH83] J. A. van Hulzen. Code Optimization of Multivariate Polynomial Schemes: A Pragmatic Approach. In J. A. van Hulzen, editor, *Computer Algebra, Eurocal 83 (European Computer Algebra Conference, London, England, March 1983*, pages 286–300, Springer-Verlag, Berlin, 1983.

[Wir80] Michael C. Wirth. *On the Automation of Computational Physics.* PhD thesis, University of California, Davis, October 1980. Also available as Lawrence Livermore National Laboratory, Livermore, Report UCRL-52996 (October 1980).